The Non-overlapping Constraint between Objects Described by Non-linear Inequalities

Ignacio Salas, Gilles Chabert, Alexandre Goldsztejn

The non-overlapping constraint between objects described by non-linear inequalities

Ignacio Salas[1], Gilles Chabert[1], and Alexandre Goldsztejn[2]

[1] Mines de Nantes - LINA (UMR 6241)
{ignacio.salas,gilles.chabert}@mines-nantes.com
[2] CNRS - LINA (UMR 6241)
alexandre.goldsztejn@univ-nantes.fr

Abstract. Packing 2D objects in a limited space is an ubiquitous problem with many academic and industrial variants. In any case, solving this problem requires the ability to determine where a first object can be placed so that it does not intersect a second, previously placed, object. This subproblem is called the non-overlapping constraint. The complexity of this non-overlapping constraint depends on the type of objects considered. It is simple in the case of rectangles. It has also been studied in the case of polygons. This paper proposes a numerical approach for the wide class of objects described by non-linear inequalities. Our goal here is to *calculate* the non-overlapping constraint, that is, to describe the set of all positions and orientations that can be assigned to the first object so that intersection with the second one is empty. This is done using a dedicated branch & prune approach. We first show that the non-overlapping constraint can be cast into a Minkowski sum, even if we take into account orientation. We derive from this an *inner contractor*, that is, an operator that removes from the current domain a subset of positions and orientations that necessarily violate the non-overlapping constraint. This inner contractor is then embedded in a *sweeping* loop, a pruning technique that was only used with discrete domains so far. We finally come up with a branch & prune algorithm that outperforms RSOLVER, a generic state-of-the-art solver for continuous quantified constraints.

1 Introduction

The goal of this article is to calculate the set of all positions and orientations that can be given to an object so that it does not overlap a second one (see the left graphic in Figure 1, which shows the simpler case where no orientation is considered). We address the general case of objects described by nonlinear inequalities. Calculating this set is a key task for solving packing problems, that consists in placing a set of objects in a bounded space so that they do not overlap pairwise.

In this introduction, we first define *objects* in our context and give a precise statement of the non-overlapping constraint. Then, we explicit the type of objects we consider and explain what we mean by *calculating* a set. We finally mention related works.

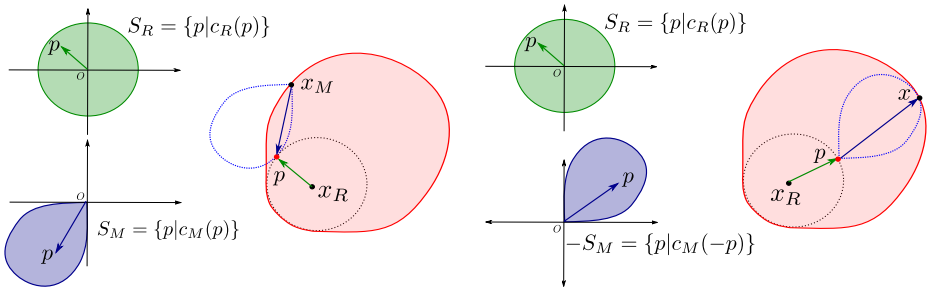

Fig. 1. Non-overlapping constraint. *Left:* two objects S_R and S_M, the region in red represents the set of all positions x_M for S_M that violate the non-overlapping constraint. *Right:* two objects S_R and $-S_M$ and their Minkowski sum, which coincides with the overlapping constraint (the negation of the non-overlapping constraint).

In Section 2, we show that our problem can be cast into the calculation of a Minkowski sum. Based on this observation, we propose a branch & bound algorithm in Section 3. Experimental results are shown in Section 4 and a conclusion follows.

1.1 Object definition

For clarity, let us first assume that the orientation is fixed, that is, objects can only be translated.

Translating an object means fixing the position of a particular point that we call the *origin*. This origin point can be arbitrarily chosen. For instance, in rectangle packing, the origin of a rectangle can be a vertex or its center point. Once this convention for the origin is made, the shape of the object is just a regular constraint. To illustrate this, let us consider again rectangle packing. If the origin is the lower-left corner, then a rectangle of dimensions l_1 and l_2 is the set of all $p \in \mathbb{R}^2$ satisfying

$$c(p) \iff 0 \leq p_1 \leq l_1 \wedge 0 \leq p_2 \leq l_2. \tag{1}$$

Alternatively,

$$c(p) \iff -\frac{l_1}{2} \leq p_1 \leq \frac{l_1}{2} \wedge -\frac{l_2}{2} \leq p_2 \leq \frac{l_2}{2}. \tag{2}$$

defines a rectangle with the center point as origin, which, of course, is just a shift of the previous constraint. So, the shape of an object can be expressed as a constraint, the latter containing an implicit convention for the origin. As a last example, a circle of radius r is the set of all $p \in \mathbb{R}^2$ such that

$$c(p) \iff \|p\| \leq r \tag{3}$$

the origin being, in this case, the center of the circle.

This definition of $c(p)$ corresponds to an object with no translation nor rotation. The general constraint corresponding to an object translated by x and rotated by α is easily obtained as follows. In the case of translation only, the part of the plane covered by an object placed at some position x is the set of all p such that $c(p-x)$ is satisfied. Let us introduce orientation. By a classical geometric argument, an object placed at x and turned by some angle α is the constraint

$$c\big(R_{-\alpha}(p-x)\big) \tag{4}$$

where R_α is the rotation matrix with angle α:

$$R_\alpha = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}. \tag{5}$$

Equivalently, we can say that $c(p) \equiv c(R_0(p-0))$ represents the object placed at 0 and rotated by the angle 0. To conclude:

- *an object* is a constraint on the plane (i.e., with two variables),
- *placing* an object means fixing the coordinate of its implicit origin,
- *orienting* an object means fixing the angle of the rotation around its implicit origin.

In this paper we consider objects described by nonlinear inequalities $c(p) \iff f(p) \leq 0$. E.g., a circle of radius 1 which origin is the center point is the set of all $p \in \mathbb{R}^2$ such that $f(p) \leq 0$ with $f : p \mapsto \|p\| - 1$. For the clarity of presentation, we will assume each object to be described by a single inequality, but our results can be easily extended to the more general case of objects described by disjunctions of conjunctions of inequalities. This is possible by introducing min and max operators. For instance, $(f_1(x) \leq 0 \vee f_2(x) \leq 0)$ is equivalent to $\max\{f_1(x), f_2(x)\} \leq 0$. Differentiability is not required by our algorithm. In fact, there is also no assumption on the functions involved, except that they are defined by mathematical expressions based on usual operators ($+$, $\times$, $\sqrt{.}$, exp, etc.). In particular, there is no convexity assumption on the input objects.

1.2 The non-overlapping constraint

We can now focus on the non-overlapping constraint. The non-overlapping constraint involves two objects, one being fixed, the other representing the unknowns of the problem. For this reason, we will call "reference object" the first one and denote by c_R its describing constraint. The second one will be called "moving object" and its constraint denoted by c_M.

We will show at the end of this section that the general case, where both objects are translated and/or rotated, can actually be obtained from the simpler case where the reference object has no transformation. Intuitively, the frame where the overlapping constraint is stated can be centered on the reference object and aligned with its orientation, albeit the exact formula is not so trivial.

So, we shall only introduce in our definition below the position x_M and the rotation angle α_M of the moving object. The non-overlapping constraint is the negation of the overlapping constraint that can be stated as follows.

Definition 1 (Overlapping constraint).
Given two constraints c_R and c_M, a vector $x_R \in \mathbb{R}^2$ and $\alpha_R \in [0, 2\pi]$:

$$\text{overlap}_{(c_R,c_M)}(x_M, \alpha_M) \iff \exists p \in \mathbb{R}^2, c_R(p) \wedge c_M\big(R_{-\alpha_M}(p - x_M)\big). \quad (6)$$

In the case of translation only, this simplifies to

$$\text{overlap}_{(c_R,c_M)}(x_M) \iff \exists p \in \mathbb{R}^2 c_R(p) \wedge c_M(p - x_M). \quad (7)$$

Our goal is to *calculate* the overlapping constraint. By *calculating*, we mean here that an explicit (numerical but verified) representation of the solution set $\mathcal{S}' := \{(x_M, \alpha_M),\ \text{overlap}_{(c_R,c_M)}(x_M, \alpha_M)\}$ (or $\mathcal{S} := \{x_M,\ \text{overlap}_{(c_R,c_M)}(x_M)\}$ in the case of translation only) has to be returned by our algorithm.

We show now that the overlapping constraint in the case where the reference object is given a position x_R and an orientation α_R can be tested using $\mathcal{S}'$ (and hence using its explicit representation). More precisely:

Proposition 1. *The moving object with position x_M and orientation α_M overlaps the reference object with position x_R and orientation α_R iff*

$$(R_{-\alpha_R}(x_M - x_R), \alpha_M - \alpha_R) \in S' \quad (8)$$

Proof. By definition, (x_M, α_M) satisfies the overlapping constraint with the reference object at position x_R and orientation α_R iff exists $\exists p \in \mathbb{R}^2$ such that $c_R(R_{-\alpha_R}(p - x_R))$ and $c_M(R_{-\alpha_M}(p - x_M))$. This is equivalent to $\exists q \in \mathbb{R}^2$, namely

$$q = R_{-\alpha_R}(p - x_R) \iff p = R_{\alpha_R} q + x_R, \quad (9)$$

such that $c_R(q)$ and

$$c_M(R_{-\alpha_M}(R_{\alpha_R} q + x_R - x_M)) \iff c_M(R_{-\alpha_M+\alpha_R}(q + R_{-\alpha_R}(x_R - x_M))). \quad (10)$$

This is equivalent to (8). ▲

Remark 1. The non-overlapping constraint is usually considered for packing applications. Its description is easily obtained from the one of the non-overlapping constraint. The former is preferred here because it simplifies the constraints expressions and its link with the Minkowski sum presented in Section 2.

1.3 Intervals, Boxes and Paving

The representation we use is called a *paving*. This representation is a natural choice in the context of constraint programming where the large majority of algorithms dedicated to continuous variables, if not to say all, assume domains of variables to be intervals (see, e.g., [BG06,JKDW01]). In the following definition, we call *box* a Cartesian product of d intervals where d is either 2 in the case of $\mathcal{S}$ or 3 in the case of $\mathcal{S}'$.

Definition 2 (Paving). *A paving of a set $\mathcal{S} \subset \mathbb{R}^d$ is a triplet $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ where $\mathcal{I}$ (for "inside"), $\mathcal{O}$ (for "outside") and $\mathcal{B}$ (for "boundary") are three sets of boxes verifying*

$$\cup\mathcal{I} \subset \mathcal{S}, \quad (\cup\mathcal{O}) \cap \mathcal{S} = \emptyset \quad \textit{and} \quad \cup(\mathcal{B} \cup \mathcal{I} \cup \mathcal{O}) = \mathbb{R}^d. \quad (11)$$

An example of paving is shown in Figure 2.

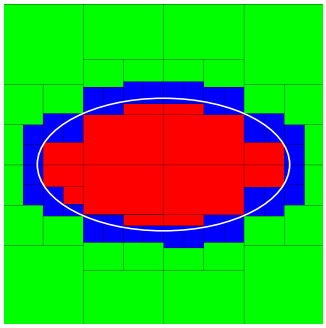

Fig. 2. Paving of an ellipsis.The interior red boxes belongs to $\mathcal{I}$, the unknown blue boxes in the boundary belongs to $\mathcal{B}$ and the green outer boxes belongs to $\mathcal{O}$.

1.4 Contribution and Related Works

This paper proposes an algorithm that computes a paving of the overlapping constraint. One contribution is on the modeling aspect of this problem: we show that the overlapping constraint can be expressed as a Minkowski sum. On the one hand, this generalizes the approach and simplifies the description of the algorithms. On the other hand, it allows handling the simple translation case and the more involved translation and rotation case homogeneously by casting the rotation to a translation into an augmented space (see Proposition 2). The other contribution is algorithmic. We propose an original *inner* contractor for this problem, that is, an operator that identifies parts of the solution set. This operator implements a *sweeping loop* and exploits the properties of the Minkowski sum. The second operator is an outer rejection test based on classical constraint propagation. Both are interleaved in a branch and prune algorithm that computes the desired paving.

From another point of view, definition 1 means that our problem falls into the category of existentially-quantified constraints. A state-of-the-art algorithm for calculating a paving with existentially-quantified inequalities is given in [Rat06] and is implemented in the RSOLVER tool [Rat] (Rsolver implements a general algorithm, somehow a numerical version of CAD, for solving quantified constraints).

General techniques [GJ06,IGJ12] for quantified *equality* constraints could also be used: either by adapting [GJ06] to compute an over approximation of the boundary of the overlapping constraint, or using a necessary condition for the boundary of the overlapping constraint expressed as an under-constrained system of equations and using [IGJ12]. However, the overlapping constraint naturally involves inequality constraints and using costly techniques dedicated to equality constraints turns out to be counterproductive.

Finally, we shall mention that an exact formula for the overlapping set $\mathcal{S}$ has been given in [BGT01] in the case where objects are polytopes. The set, in this

case, is the convex hull of the points obtained by summing one vertex of the first polytope to one vertex of the second one.

2 Overlapping as a Minkowski Sum

In this section, we show that the overlapping constraint can be reformulated as a Minkowski sum. This relation underlies our branch & bound solver, that will be presented further. Let us first recall the definition of the Minkowski sum of two sets:

Definition 3 (Minkowski sum).
Given two equi-dimensional sets $S_1, S_2 \subseteq \mathbb{R}^d$, the Minkowski sum is

$$S_1 + S_2 = \{x_1 + x_2 \in \mathbb{R}^d : \ x_1 \in S_1, \ x_2 \in S_2\}. \tag{12}$$

The Minkowski difference is defined accordingly by:

$$S_1 - S_2 = \{x_1 - x_2 \in \mathbb{R}^d : \ x_1 \in S_1, \ x_2 \in S_2\}. \tag{13}$$

The right graphic of Figure 1 (page 2) shows an example of two sets and their Minkowski sum.

Considering S_1 as a constraint c_1 (i.e., $x \in S_1 \iff c_1(x)$) and, similarly, S_2 as c_2, we have, equivalently:

$$S_1 + S_2 = \{x \in \mathbb{R}^d : \ \exists p \in \mathbb{R}^d, \ c_1(p) \wedge c_2(x - p)\}. \tag{14}$$

where d is the number of variables in the constraints. Comparing (14) and (7), we immediately see that $\mathcal{S} = S_R - S_M$, i.e. the overlapping constraint can be represented as a Minkowski sum in the case of translation only.

We show now that $\mathcal{S}'$ is also a Minkowski sum, a less trivial result. To this end, we embed the moving object $S_M \subseteq \mathbb{R}^2$ into $\mathbb{R}^3$ encoding its rotation within the additional dimension:

$$\mathcal{S}'_M := \{(v, \beta) : c_M(R_\beta\, v)\} = \{(v, \beta) : R_\beta\, v \in \mathcal{S}_M\}. \tag{15}$$

Now, the following proposition states that the overlapping constraint with rotation $\mathcal{S}'$ can be written as a Minkowski difference of two such "augmented" sets (see Figure 3).

Proposition 2.

$$\mathcal{S}' = S_R \times \{0\} - S'_M. \tag{16}$$

Proof. By definition, $(x_M, \alpha_M) \in \mathcal{S}'$ holds if and only if $\exists p \in \mathbb{R}^2$ such that $c_R(p)$ and $c_M(R_{-\alpha_M}(p - x_M))$. Equivalently, there exists $u_R \in \mathcal{S}_R$ and $u_M \in \mathcal{S}_M$ such that $u_R = p$ and $u_M = R_{-\alpha_M}(u_R - x_M) \iff x_M = u_R - R_{\alpha_M} u_M$. Finally, the vector (x_M, α_M) is proved to be the sum of $(u_R, 0) \in \mathcal{S}_R \times \{0\}$ and $(R_{\alpha_M} u_M, \alpha_M) \in \mathcal{S}'_M$. ▲

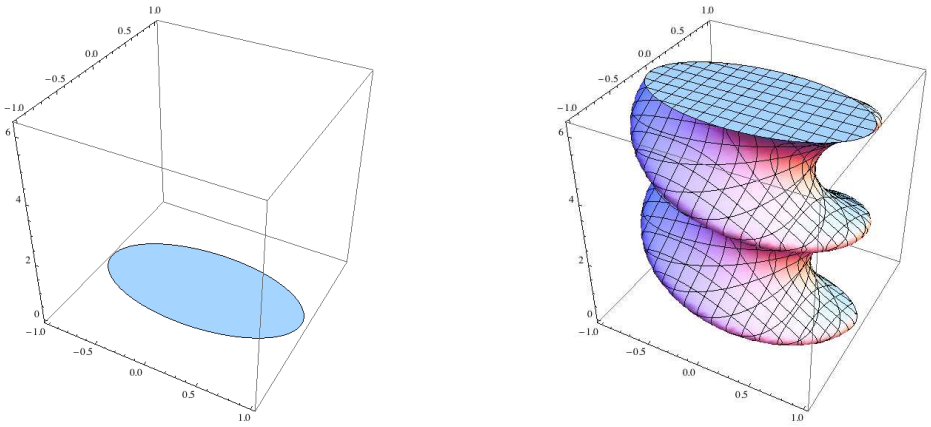

Fig. 3. Sets whose Minkowski difference gives the overlapping constraint of two ellipsis, when rotation is taken into account. *Left:* the augmented reference ellipsis, with rotation coordinate set to 0. *Right*: the augmented moving ellipsis S'_M.

3 Algorithm

From now on, our goal is to calculate a paving $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ of the sum $\mathcal{S}$ of two sets $\mathcal{S}_1$ and $\mathcal{S}_2$. According to the previous section, the link with the non-overlapping constraint is made by setting $\mathcal{S}_1$ to either S_R or S'_R and $\mathcal{S}_2$ to either $-S_M$ or $-S'_M$.

Algorithm 1 below is based on a classical SIVIA-like branch & contract recursive loop [JW93,CJ09]. The core operation made on a box $[x]$ can be broken into three steps. First, $[x]$ is contracted to a box $[x]'$ by an *inner contractor* C_{in}, that is, an operator that guarantees:

$$[x]' \subseteq [x] \ \wedge \ [x] \backslash [x]' \subseteq \mathcal{I}. \tag{17}$$

If $[x]' \neq \emptyset$ then $[x]'$ is contracted to a box $[x]''$ by an *outer contractor* C_{out} that guarantees:

$$[x]'' \subseteq [x]' \ \wedge \ [x]' \backslash [x]'' \subseteq \mathcal{O}. \tag{18}$$

Finally, if $[x]'' \neq \emptyset$ then $[x]''$ is bisected in two new boxes that are pushed in the list of boundary boxes $\mathcal{B}$. The recursion stops when the total surface of boxes in $\mathcal{B}$ is less than $\varepsilon\%$ of the initial box surface (s_0), ε being a user-defined parameter.

The originality of our approach lies in the inner contractor that we describe now. It is dedicated to the handled problem as it makes use of the specific structure of the quantified constraint (14).

3.1 Inner contractor

Our inner contractor is based on the *inner arithmetic* and a *sweep* loop: The former builds small inner boxes, and the latter makes the union of these boxes in order to remove slices of the initial box, leading to a so called *inner contraction*.

Algorithm 1: $(\mathcal{I}, \mathcal{B}, \mathcal{O}) = \textbf{pave}([x], \varepsilon)$

```
1  s_0 ← surface([x]);                                          // initial surface
2  I ← ∅;   O ← ∅;   B ← {[x]};
3  s ← s_0;                                                     // current surface of B
4  while (s > ε × s_0) do
5      [x] ← box of B with the largest surface;    // (use a heap structure for that)
6      pop [x] from B;
7      s ← s− surface([x]);                                     // update the surface
8      [x]' ← C_in([x]);   I ← I ∪ ([x]\[x]');                  // inner contraction
9      [x]'' ← C_out([x]');   O ← O ∪ ([x]'\[x]'');             // outer contraction
10     if ([x]'' ≠ ∅) then
11         ([x]_1, [x]_2) ← bisect([x]'');
12         push [x]_1 and [x]_2 in B;
13         s ← s+surface([x]_1)+surface([x]_2);                 // update the surface
14     end
15 end
16 return (I, B, O);
```

The inner arithmetic is a variant of the classical interval arithmetic that allows to build a sub-box of a box $[x]$ that is inside a given set $\mathcal{S}$ described by inequalities. This technique was first introduced in §3 of [CB10] and used in [ATNC14] in the context of global optimization. This inner arithmetic can also be used with a initial point (or initial box) that is *inflated*, that is to say, given a box $[x]$ and $\tilde{x} \in [x]$, it produces a box $[\tilde{x}]$ such that

$$\tilde{x} \in [\tilde{x}] \subseteq [x] \ \wedge \ [\tilde{x}] \subseteq \mathcal{S}, \tag{19}$$

or an empty box if $\tilde{x} \notin \mathcal{S}$. This arithmetic has similar properties to its classical counterpart: the time complexity is in the length of the constraint expression and gives an optimal box $[\tilde{x}]$ (i.e., of maximal size in every dimension) if no variable occurs twice in the expression.

Before describing how to contract a box $[x]$ with this new arithmetic, let us first address a simpler question: how to find a subbox of $[x]$ that is inside $\mathcal{S}$?

A possible answer is to look for two boxes $[x]_1$ and $[x]_2$ such that

$$[x]_1 \subseteq \mathcal{S}_1, \quad [x]_2 \subseteq \mathcal{S}_2 \ \text{ and } \ ([x]_1 + [x]_2) \cap [x] \neq \emptyset \tag{20}$$

because, in this case, $([x]_1 + [x]_2) \subseteq [x] \cap \mathcal{S}$. To find such boxes, one can calculate in parallel two pavings, one of $\mathcal{S}_1$ and one of $\mathcal{S}_2$, and stop the process as soon as two boxes satisfying (20) are found. By combining boxes of the first paving with boxes of the second one, this approach amounts to run a branch & bound in a $(2 \times d)$-dimensional space. Note that this branch & bound is a sub-solver embedded in the main one, the one for the x variable. Our goal is to reduce the sub-solver to d dimensions only, which is, by the way, the incompressible price to pay for handling d existentially-quantified parameters.

To this end, we use the same idea as above, but this time based on Relation (14) (see Figure 4). To build an inner box in $[x]$, let us first assume that we

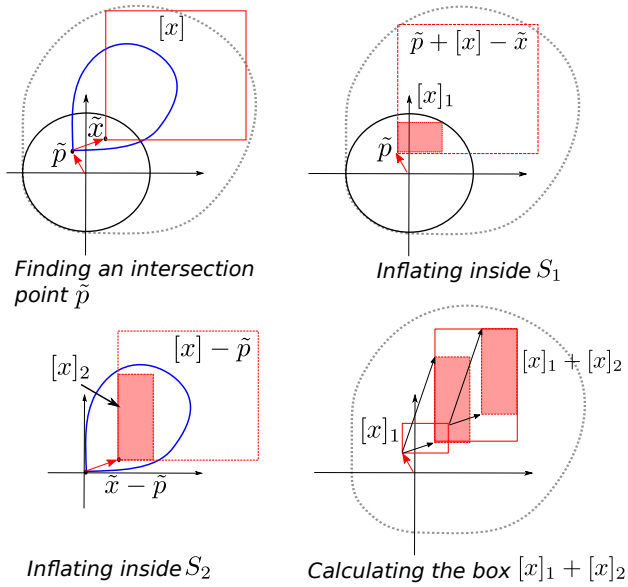

Fig. 4. Steps of the inner Contractor.

have picked some point $\tilde{x}$ inside $[x]$ (this point is, in fact, automatically yield by the sweep loop, as this will be explained in Figure 5). Then we look for another point $\tilde{p}$ such that

$$c_1(\tilde{p}) \wedge c_2(\tilde{x}-\tilde{p}). \tag{21}$$

Finding this point is the task of the subsolver.[3]

Once $\tilde{p}$ is found, it is "inflated" to a subbox $[x]_1$ of $(\tilde{p}+[x]-\tilde{x})$ that is inside $\mathcal{S}_1$, which is possible with the inner arithmetic. The point $(\tilde{x}-\tilde{p})$ is also inflated to a sub-box $[x]_2$ of $([x]-\tilde{p})$ that is inside $\mathcal{S}_2$. However, if $\tilde{x}$ turns out to be outside $\mathcal{S}$, the last inflation cannot succeed and the process is interrupted in this case. Otherwise, the resulting box satisfies $([x]_1+[x]_2) \subseteq \mathcal{S}$ and $([x]_1+[x]_2) \cap [x] \neq \emptyset$.

Note that $([x]_1 + [x]_2) \cap [x] \neq \emptyset$ is just a consequence of $\tilde{x} \in [x]$. So the initial boxes used for both inflations are somehow arbitrary, but fixing them as we did is an heuristic that tends to maximize the surface of the final inner box $([x]_1 + [x]_2) \cap [x]$.

[3] This subsolver is implemented with a standard branch & prune based on HC4 [BG06]. Since only one solution is sought, at each node of the search, we check inequalities with a point $\tilde{p}$ picked randomly in the current domain. If both are satisfied, the search is interrupted and $\tilde{p}$ is returned. The depth of the search is also controlled by a precision on the domain width, which ensures that the subsolver terminates in bounded time. In case of normal termination, no $\tilde{p}$ hence no inner box has been found.

Now that we have a technique to build an inner box inside $[x]$ that contains a specific point $\tilde{x}$, we can use this service inside a *sweep* loop. The sweep loop can be simply viewed as a way to contract a box by "piling up" boxes until some face is entirely covered. This is quickly depicted in Figure 5. The interested reader may refer to [CB10] for further details.

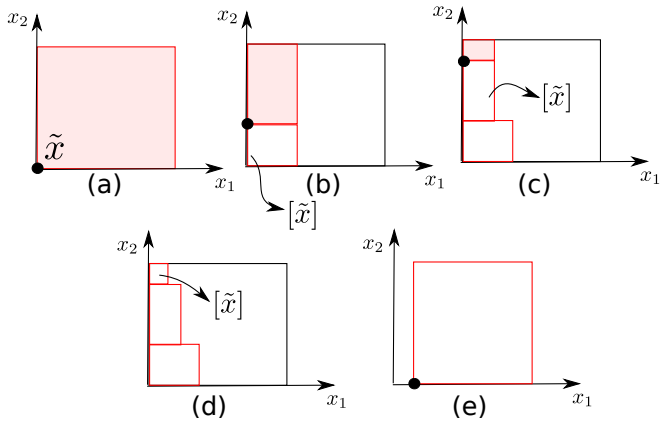

Fig. 5. The sweep loop. The sequence of pictures illustrates a contraction for the lower bound of x_1. At each step, the point $\tilde{x}$ to be inflated is the lower-left corner of the gray box. The inner box $[\tilde{\mathbf{x}}]$ is painted in white. The lower bound of x_1 can be reduced as soon as the projection of the white boxes on x_2 spans the face $[x_2]$, which is the case at step **e)**.

3.2 Outer contractor

The outer contractor is less sophisticated than the inner one and acts as a simple *rejection test*: the box is either entirely discarded or kept intact.

Rejecting a box $[x]$ means proving $[x] \not\subseteq S_1 + S_2$, that is

$$\forall x \in [x] \quad \forall p \in \mathbb{R}^d,\ \neg(c_1(p) \wedge c_2(x-p)). \tag{22}$$

This assertion can be checked by running the same subsolver we used for the inner contractor, except that the point $\tilde{x}$ is replaced by the current box $[x]$. If the subsolver finds no solution, the previous assertion is proven. Note that only the coordinates of p are bisected, so the subsolver actually proves a stronger assertion if the contraction with respect to c_2 is not optimal (the actual assertion depends on the consistency level enforced by the contraction with c_2). Note also that the precision used in the subsolver is dynamically set to the width of $[x]$ in order to have a somewhat uniform time spent by the subsolver throughout the global search (on x). This dynamic precision also ensures that the outer contractor is *convergent*, that is, it rejects any small enough boxes outside $\mathcal{S}$.

One may be surprised by the simplicity of this rejection test and expect a more elaborated contractor for the outer region, inspired by what we did for the inner region. But the situation could be interpreted in the other way around. Since the overlapping constraint is in two dimensions only, an inner satisfiability test would probably fits, as long as it is fast and convergent. However, such a test amounts to prove for $[x] \subseteq S_1 + S_2$ the following assertion

$$\forall x \in [x] \quad \exists p \in \mathbb{R}^d,\ (c_1(p) \wedge c_2(x-p)).$$

and, contrary to (22), the $\forall$ and $\exists$ quantifiers are involved, which means that the problem is much harder. So, the inner contractor can be seen here as a way to make up for the lack of inner test.

Our previous argument is only based on running time. It is clear that an outer contractor could also lead to a more compact paving, but the size of the paving is anyway conditioned by the representation of the boundary so that a drastic gain on this aspect is not really expectable.

4 Experimental results

Experimental setup

The algorithm proposed in this article calculates a paving $(\mathcal{I}, \mathcal{B}, \mathcal{O})$ of the overlapping constraint. The difficulty of this task mainly depends on three criteria:

- *variable occurrences*: the number of times each variable appears in the expressions of the inequalities directly affects the efficiency of the contractors; the more occurrences a variable, the less efficient contractors. This is a well-known drawback of the classical interval arithmetic that carries over the inner arithmetic (used by the inner contractor).
- *convexity*: if objects are non-convex, the boundary of the non-overlapping constraint will be less smooth. So the paving will be more complicated (hence more time consuming), especially near the boundary.
- *degrees of freedom*: that is, whether we take into account rotation or not. Allowing rotation gives a problem of much higher difficulty for multiple reasons. First, the size of the paving is exponential in the number of degrees of freedom so we cannot expect to get a 3D paving within the time scale of a 2D paving. Second, the angle in the inequalities creates a lot of multi-occurrences (see Equations (4) and (5)) and considerably increase non-convexity by the introduction of trigonometric functions.

Our benchmark is based on these criteria. We have made two types of experiments. The first one is with translation only. We have considered three objects of increasing difficulty. Object №1 is a simple ellipsis:

$$\text{Object №1}: \quad (p_1/2)^2 + p_2^2 \leq 1. \tag{23}$$

Object №2 is an ellipsis rotated by some fixed angle. Objects №1 and №2 are obviously of equal complexity if rotation is a degree of freedom, but not if

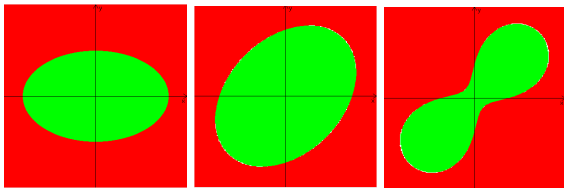

Fig. 6. Objects of increasing complexity. *From left to right*: objects №1, 2 and 3.

we limit ourselves to translation. This is because the rotated object introduces multi-occurrences for p_1 and p_2:

$$\text{Object №2}: \quad 1.5 \times p_1^2 + 1.5 \times p_2^2 - p_1 \times p_2 - 0.2 \leq 0. \tag{24}$$

Finally, the third object has a "peanut" shape. It cumulates multiple occurrences and non-convexity, as depicted in Figure 6:

$$\text{Object} \quad \text{№3}: \quad (p_1^2 + p_2^2)^2 - 2 \times (p_1 \times p_2) - 0.02 \leq 0. \tag{25}$$

The non-overlapping constraint involves two objects: the "reference" one (which coordinates are fixed at the origin of the frame) and the "moving" one that represents the unknowns. We have considered all possible combinations with the three types of objects above, that is, the 6 first cases in Table 1.

Case	Reference object	Moving object	Rotation
1	1	1	no
2	1	2	no
3	1	3	no
4	2	2	no
5	2	3	no
6	3	3	no
7	1	1	yes
8	3	3	yes

Table 1. Cases of study.

In the second set of experiments, we have introduced rotation and tested with two ellipsis and two "peanuts" (cases 7 and 8).

In each experiment, the paving process is interrupted when the total surface of the boundary $\mathcal{B}$ is less than ε% the surface of the initial box (initial domain for the variables), where ε is a user-defined precision parameter. We have applied the same policy with RSOLVER, the tool we are comparing to.

Since the precision is in proportion of the initial domain surface, it should be noted that the quality of the paving depends also on the width of the initial

enclosure. The larger the initial domain, the less precise the resulting paving. For this reason, to give ε a meaningful value, we have set in the experiments the initial box to a fairly accurate enclosure of the overlapping set $\mathcal{S}$ or $\mathcal{S}'$ (as it can be seen in Figure 7 and subsequent).

Results (without rotation)

We first compare in Table 2 the running times obtained by RSOLVER and our algorithm for the 6 first cases, with a precision ε set each time to 3.25%. This choice for ε corresponds to the minimal value that gives no timeout with RSOLVER. The pavings obtained are depicted in Figure 7.

We then provide in Figures 8 a more detailed analysis for the two extreme cases (case 1 and 6), where ε varies from 10% downto 1%. They show some significant absolute performance gain, as well as some better asymptotic behavior with respect to RSolver.

Case	Rsolver	Our algorithm
1	4,07	0,37
2	51,55	2,67
3	611,85	7,90
4	132,46	5,58
5	656,11	12,00
6	771,00	26,82

Table 2. Running time (in s) for the 6 first cases (the precision is set to 3.25%).

The results first confirm the presumed complexity levels of the different cases, since the "harder" instances indeed require more time to be solved. They also show that our algorithm is more competitive than RSOLVER. But, of course, RSOLVER is a generic solver that does not take advantage of the specific structure of the handled problem. It should also be noted from the graphics that the gap between our approach and RSolver, in a given case, increases as we use smaller values for the precision.

Results (with rotation)

We only present here preliminary results with rotation.

Figures 9 and 10 show 2D sections of the 3D pavings we have obtained in cases 7 and 8 with a precision set to 3.25%. A 2D section is obtained by fixing the angle to some value and selecting the boxes in the 3D pavings for which the dimension of the angle contains this value. The two other dimensions are plotted.

We have set the domain of the angle to the interval $[0, 0.7]$ for the case 7, and to the interval $[0, 0.3]$ for the case 8. The only paving that was possible to

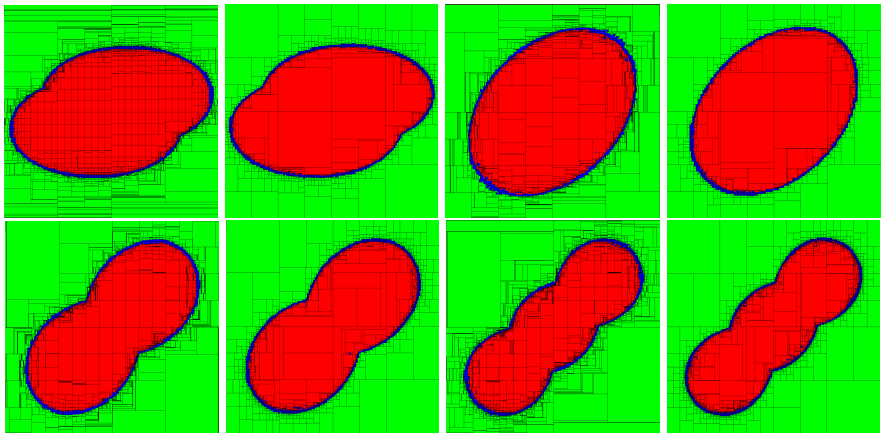

Fig. 7. Pavings obtained for cases 3-6. The precision is set to 3.25%. *Left:* with RSolver. *Right:* with our algorithm.

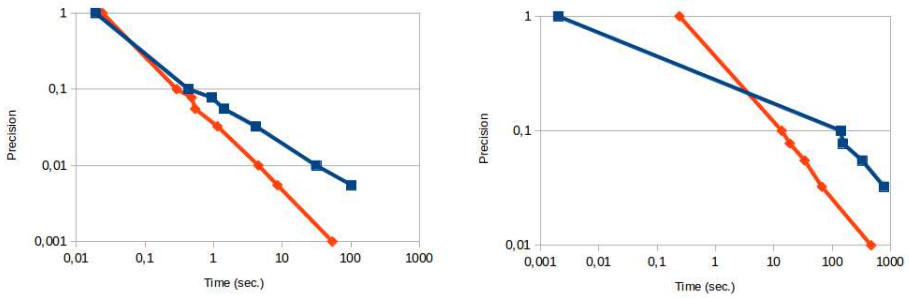

Fig. 8. Time vs Precision (left: case 1; right: case 6). Each curve represents the paving time (vertical axis) with respect to the precision ε (horizontal axis). Both axis are in logarithmic scale. The blue curve is RSolver and the red curve is our method.

obtain within the time limit was with our algorithm and for the case 7. This paving has been calculated in 9 minutes whereas RSolver does not return after 80 minutes. Both algorithm do not terminate after 100 minutes in the case 8.

When a program timeouts, only a partial result is displayed. A partial result means that the surface of $\mathcal{B}$ exceeds ε% the initial width.

The main purpose of this experiment is to show that our approach is still valid in the case where rotations are taken into account. Rotations only transform the expressions that describes the objects, without requiring any change in the algorithm itself. It is clear however that calculating a full 3D paving is a heavy task, whatever the algorithm is.

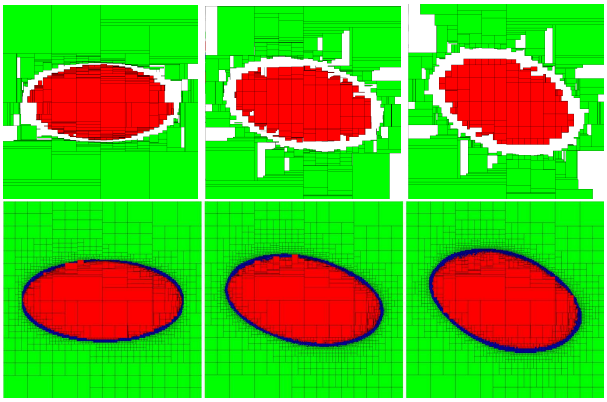

Fig. 9. 2D section of the 3D paving obtained for the case 7. *From left to right:* the rotation angle are 0, 0.4 and 0.7. *Top:* with RSOLVER. *Bottom:* with our algorithm.

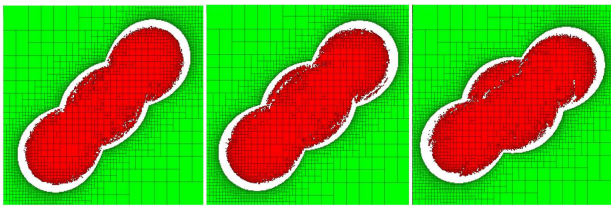

Fig. 10. 2D section of the 3D paving obtained for the case 8. From left to right, the rotation angle are 0, 0.1 and 0.3. This paving is calculated with our algorithm.

5 Conclusion

In the case of objects defined by non-linear inequalities, the non-overlapping constraint can only be handled numerically. In this paper, we have given an efficient way to generate verified pavings approximations for this constraint. These pavings represent explicitly the constraint, that is, the set of all acceptable positions and orientations of an object with respect to another. Our preliminary experiments have shown strong efficiency gains with respect to the state of the art solver for quantified numerical constraints RSolver, in particular in the case where the orientation of the objects is taken into account. In our future work, these pre-computed pavings will be incorporated within a packing algorithm, as explicit descriptions of the overlapping constraints. However, with rotation, we have seen that full 3D pavings are clearly too big so that a more adaptative approach will probably have to be considered as well. The idea would be to calculate on-the-fly sub-sets of the overlapping constraint, depending on the actual domains of the objects positions assigned by the packing solver.

References

[ATNC14] I. Araya, G. Trombettoni, B. Neveu, and G. Chabert. Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. *Journal of Global Optimization*, page (to appear), 2014.

[BG06] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In *Handbook of Constraint Programming*, chapter 16, pages 571–604. Elsevier, 2006.

[BGT01] N. Beldiceanu, Q. Guo, and S. Thiel. Non-Overlapping Constraints between Convex Polytopes. In *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*, volume 2239 of *Lecture Notes in Computer Science*, pages 392–407. Springer-Verlag, 2001.

[CB10] G. Chabert and N. Beldiceanu. Sweeping with Continous Domains. In D. Cohen, editor, *16th International Conference on Principles and Practice of Constraint Programming (CP'10)*, volume 6308 of *Lecture Notes in Computer Science*, pages 137–151, St Andrews, Scotland, 2010. Springer-Verlag.

[CJ09] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.

[GJ06] A. Goldsztejn and L. Jaulin. Inner and Outer Approximations of Existentially Quantified Equality Constraints. In *CP*, pages 198–212. Springer, 2006.

[IGJ12] D. Ishii, A. Goldsztejn, and C. Jermann. Interval-Based Projection Method for Under-Constrained Numerical Systems. *Constraints*, 17(4):432–460, 2012.

[JKDW01] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[JW93] L. Jaulin and E. Walter. Set Inversion via Interval Analysis for Nonlinear Bounded-Error Estimation. *Automatica*, 29(4):1053–1064, 1993.

[Rat] S. Ratschan. RSolver.

[Rat06] S. Ratschan. Efficient Solving of Quantified Inequality Constraints over the Real Numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.